Preliminary solution proposal, DALG part

DALG assignment #1:

1. `SkipListDeleteMin ( SkipList L ) : SkipListItem`
   ```
   \{
   P ← Header(L) // the artificial header element
   M ← P.next[0]; // first element has a minimum key
   \textbf{for} i \textbf{from} 0 \textbf{to} height(M) − 1 \textbf{do}
   \hspace{1em} P.next[i] ← M.next[i]
   \textbf{return} M
   \}
   ```

2. \( \Theta(H) \), which is supposed to be \( \Theta(\log N) \)

3. \( h \) next-pointer updates (# iterations of the for loop)

4. An item has height \( h \) with probability \( 2^{-h} \). Deleting an item of height \( h \) requires \( h \) next-pointer updates.
   
   hence expected # pointer updates = \( 1 \cdot 2^{-1} + 2 \cdot 2^{-2} + \ldots \leq \sum_{i=0}^{H} i2^{-i} \leq 2 \)

   because with \( \sum_{k=0}^{\infty} kx^k = x/(1 - x)^2 \) and \( x = 1/2 \) we have \( \sum_{i=0}^{\infty} i2^{-i} \leq 2 \).

   Hence, the average case time complexity of `SkipListDeleteMin` is \( \Theta(1) \).

5. The easiest way is to maintain an additional pointer \( \text{last} \) to the last element in the skip list, which contains a maximum key because the skip list is sorted in increasing order of keys. `FindMin` then just returns that pointer, which obviously takes constant time.

   The operation `Insert(i)` must be modified such that `last` is updated to point to \( i \) if \( i \) happens to end up as the last (w.r.t. `next[0]`-linkage) element in the skip list. Note that this also covers the case where the list was empty.

   `Delete` must update `last` if the element being deleted is the one pointed to by `last`. In that case, `last` is set to its predecessor wrt. `next[0]` linkage, which is one of the flagged items in the deletion process.

   `DeleteMin` needs modify `last` only if the list contained only a single element.

   `LookUp` needs not be modified as it does not change the list.
**DALG assignment #2:**

(a) array representation: [2, 3, 5, 7, 11, 13, 17, 19], tree representation

(b) after DeleteMin: [3, 7, 5, 19, 11, 13, 17] plus tree representation

(c) after Insert(1): [1, 2, 5, 3, 11, 13, 17, 19, 7] plus tree representation

**DALG assignment #3:**

The hash table contains the entries [0, 1, 12, 11, 14, 22, 29, 7, empty, 9], Lookup(29) probes 8 times.

**DALG assignment #4:**

(a) We apply heap sort. As the first \( n - \sqrt{n} \) elements are already sorted, we can interpret these first \( n - \sqrt{n} \) table entries as a heap. Now we insert the last \( \lceil \sqrt{n} \rceil \) elements in the heap one by another, each of which takes time \( O(\log n) \) as the heap has \( \leq n \) elements in all cases. Hence, the total worst-case time complexity is \( \lceil \sqrt{n} \rceil \cdot O(\log n) \in O(n) \).

(b) We have to find a function \( f(n) \) such that \( f(n) \cdot \log n \in O(n) \). \( f(n) \) is as big as possible if \( f(n) \cdot \log n \in \Theta(n) \), i.e., if \( f(n) \in \Theta(n / \log n) \). Hence, \( f(n) = n / \log n \) is such a solution, and we have \( \sqrt{n} \in o(n / \log n) \) because \( \log n \) grows asymptotically slower than \( \sqrt{n} \).

**DALG assignment #5:**

(a) no. There are two unary nodes.

(b) no. Level 3 is not filled completely, neither is the last level filled properly from left to right.

(c) no. The node with key 56 is out of balance (-2).

(d) preorder: \( \langle 34, 22, 15, 17, 32, 56, 47, 43, 51, 52, 66 \rangle \)

   inorder: \( \langle 15, 17, 22, 32, 34, 43, 47, 51, 52, 56, 66 \rangle \)

   postorder: \( \langle 17, 15, 32, 22, 43, 52, 51, 47, 66, 56, 34 \rangle \)